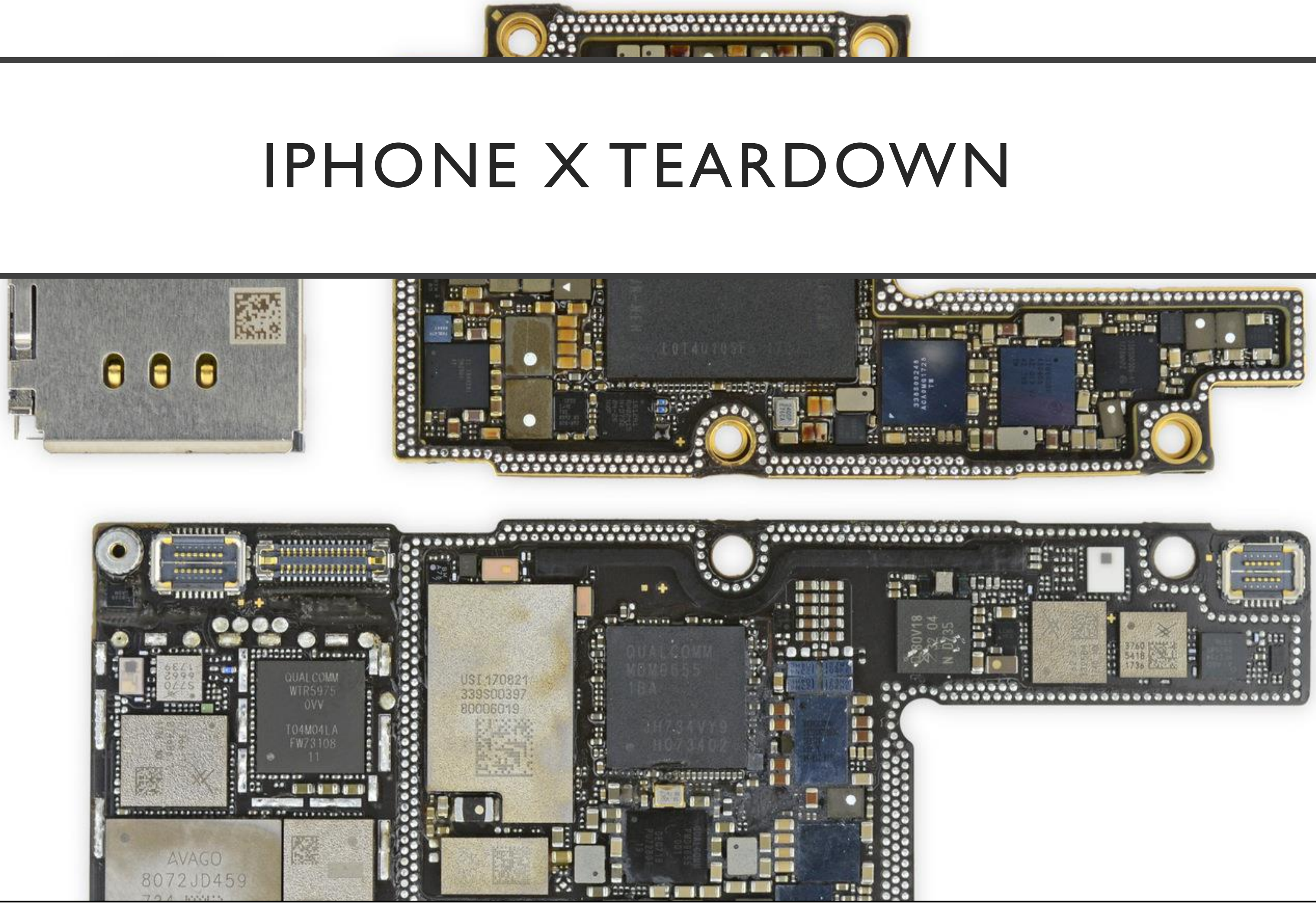
L2: EMBEDDED SYSTEM ARCHITECTURE

ESE516: IoT Edge Computing

Monday, January 28, 2019

Eduardo Garcia- edgarc@seas.upenn.edu

IPHONE X TEARDOWN



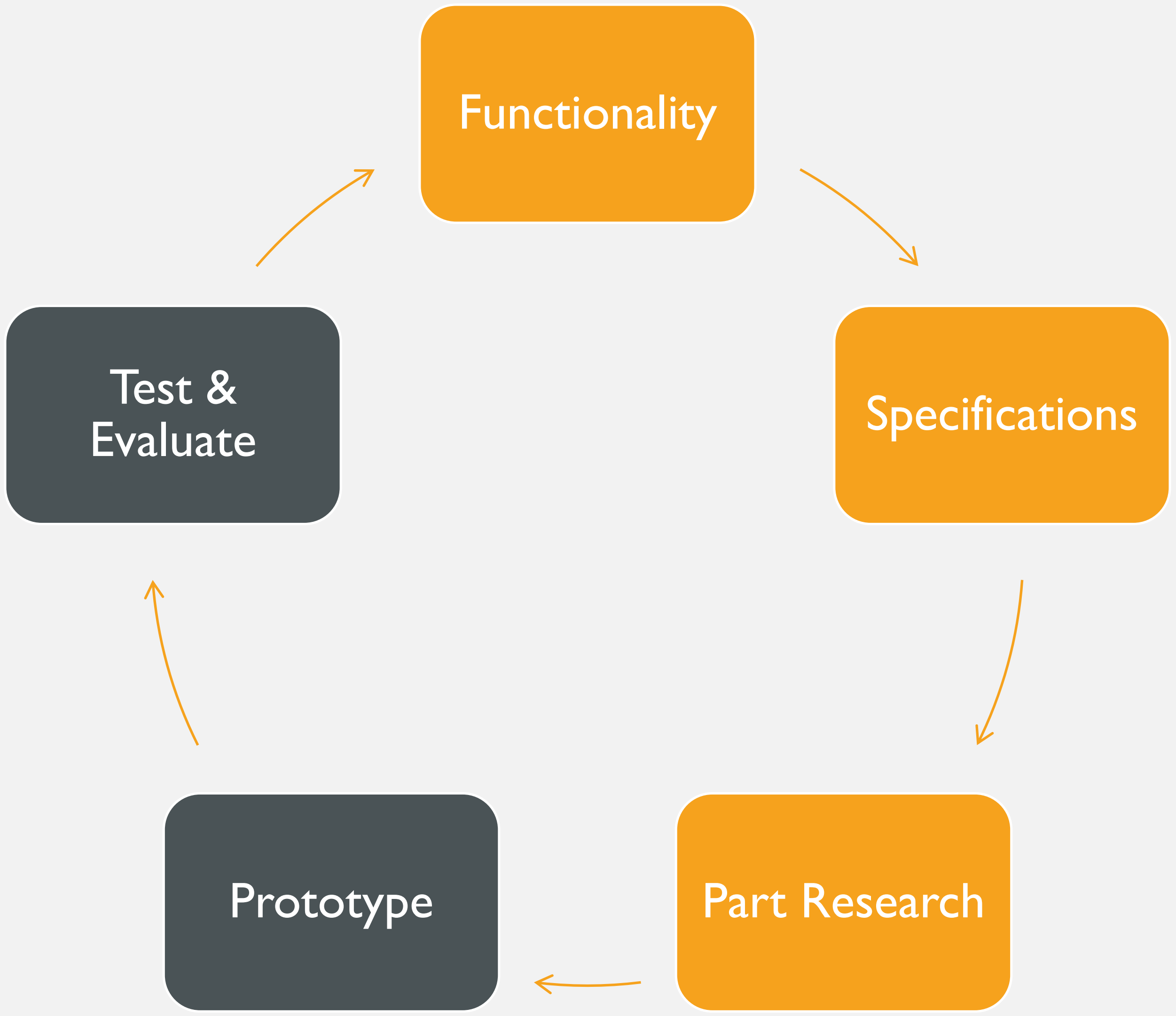
https://www.ifixit.com/Teardown/iPhone+X+Teardown/98975

CLERICAL WORK

ADMINISTRATION

- A0 is graded – please see comments on Canvas!
- Piazza posts
- Syllabus online
- A1 will be posted today – Due on **Monday February 4 2019th on midnight minus 1 minute**
- **Office hours to be posted soon**

PRODUCT DESIGN CYCLE*



DETOUR INTO EMBEDDED FIRMWARE DESIGN - CONCURRENCY

BEFORE BEGINNING…

- We have to understand the capacity of the device
- We have to understand the problem
- We have to understand how the capacity of the microcontroller can help us solve the problem

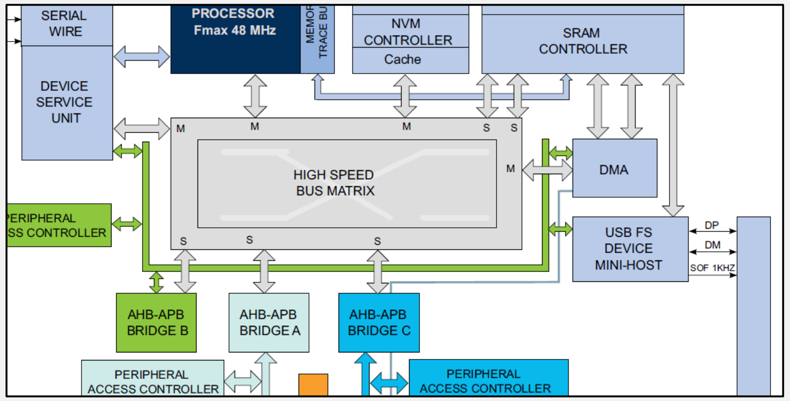
FUNCTIONS IN ASF

WAIT / BLOCKING

- Just like it sounds – the processor completes this task first before doing anything else.
- Easiest to set up

JOB / INTERRUPTS

- Non-blocking function call – so the processor can do other things until the interrupt is called.



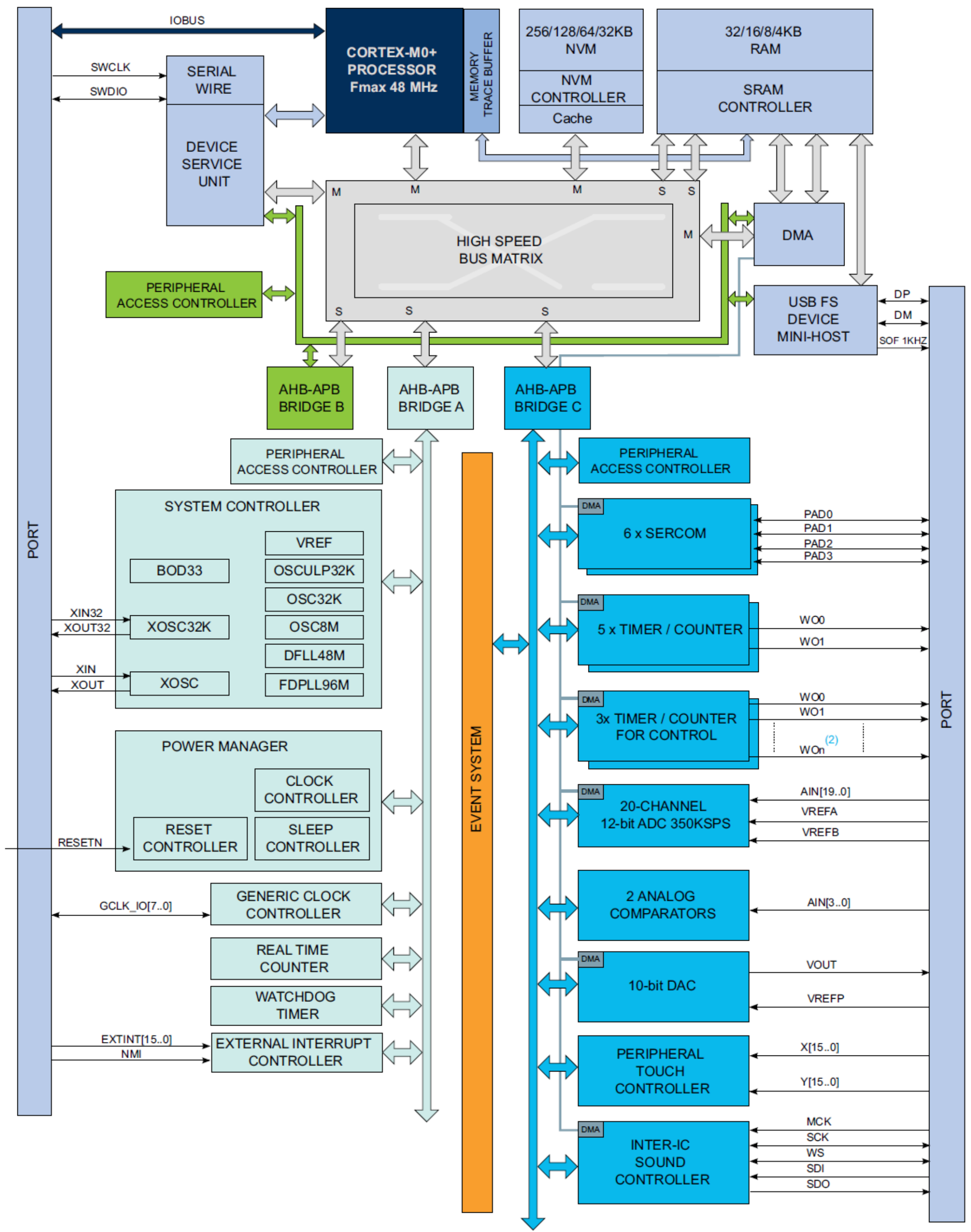
DIRECT MEMORY ACCESS (DMA)

- Links submodules together – so, for example, you could pull in data from one SPI SERCOM and send it out to another.
- The least amount of CPU interaction, hardest to set up

ARM

- 32-bit bit MCUs
- RISC – Reduced Instruction Set Computing
    - Small set of simple & general instructions
    - Typically fewer transistors than CISCs, better $, heat, power
- Family of RISC architectures developed by ARM Holdings
    - IP licensed out

Block Diagram



QUESTION FOR FIRMWARE DEVELOPMENT

- We want the product to:
  - Performs multiple operations at once
  - Offer little to no delay to the inputs of the user (and outputs to the user)
  - Not fail catastrophically attempting to do one task

QUESTION FOR FIRMWARE DEVELOPMENT

- How can we do this with an MCU that has only one core?

A: There are multiple ways to design our firmware in such a way that we offer the user parallel processes happening at once, even if there is no real concurrency happening on the MCU.

  - Use a Real Time Operating system, which handles multiple processes (threads) concurrently, switching between them

  - Programming a "bare metal" design that does executes tasks concurrently as they are available (Round Robin)

ROUND ROBIN

- Tasks are executed one after another
- No preemption – No way to exit or enter one task until the other one has endes
- Takes us close to a good concurrency model, but no quite there yet!
- Good for small ideas and prototypes, but not for a product!

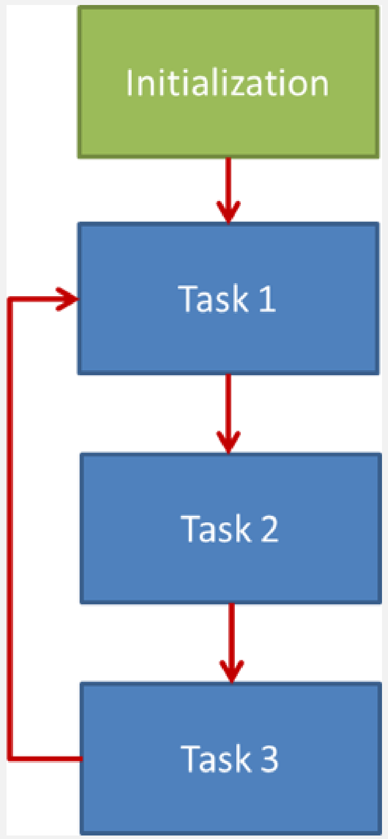


*Image taken from https://www.embeddedrelated.com/showarticle/969.php

ROUND ROBIN WITH INTERRUPTS

- Uses interrupts from peripherals to indicate when we should perform a task or not
- Better concurrency – We only execute tasks when we have data available or when we have to.
- Good enough for this class!

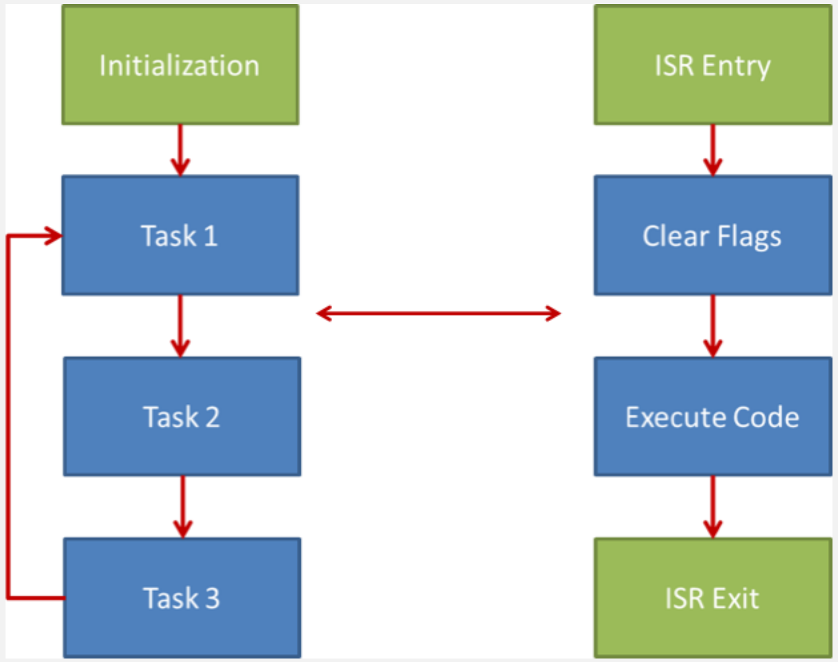


*Image taken from https://www.embeddedrelated.com/showarticle/969.php

EMBEDDED FIRMWARE DESIGN HOW DO WE ACHIEVE CONCURRENCY

HOW TO ACHIEVE CONCURRENCY

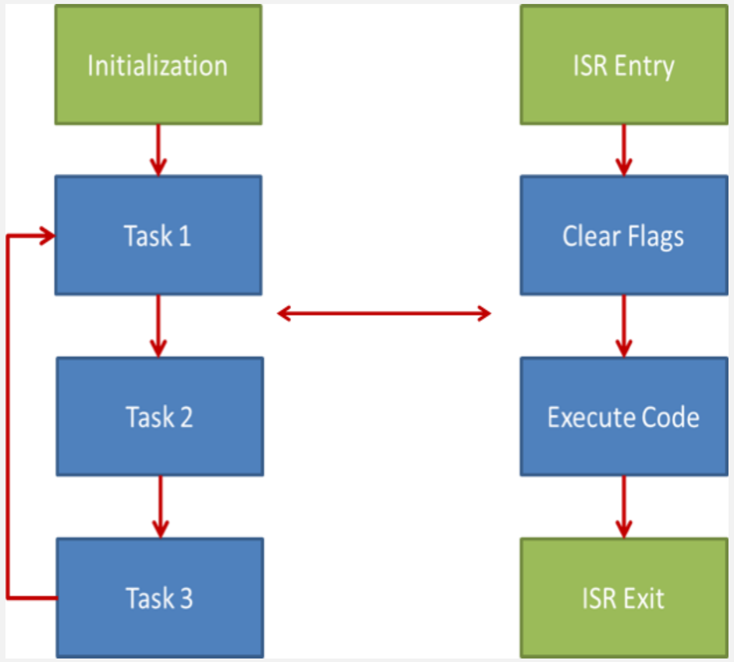
Main code executes tasks in succession, usually in order of importance

-These tasks are usually sub-state machines

The tasks can be skipped or not depending on flags

-Flags are a Boolean that indicates to the program if an event and/or data is present for the task to execute. For example, if we got UART data from the user we could set a flag that indicates to the user to run the CLI task!

Good architecture and code practices spell



HOW DOES THIS LOOK USUALLY ON CODE?

```
int main (void)
{
	//Board Initialization -- Code that initializes the HW and happens only once
	system_init();
	InitializeSerialConsole();

	//Main program loop – Each tasks is its own internal state machine
	int error = 0;
	while(1)
	{
		GetSensorData();  //Uses I2C interrupts to wait until data is received – returns if nothing to do
		ManageActuators(); //Returns inmediately if it has to do nothing to do
		ManageUI(); //Executes only every 50ms, returns if it is not its time to execute
		ManageCLI(); //Execute only if there are characters received by the user

	}
```

HOW YOU ARCHITECT YOUR CODE CAN MAKE OR BREAK YOUR PRODUCT

CODE TIP # 1 – GOOD DOCUMENTATION

CODE TIP # 1 – GOOD DOCUMENTATION

Embedded C code can be obscure and enigmatic

- Can run on 8 bit, 16 bit or 32 bit processors with their own quirks
- Can interface with thousands of external Ics, each with their own quirks
- Can have legacy code made in the 70's, with its own quirks
- Can have gone through multiple developers, each one adding quirks
- Etc!

- Clear, concise documentation is essential!

CODE TIP # 1 – GOOD DOCUMENTATION

Doxygen is a tool that allow you to make good comments directly on your code.

At the end of your project, you "compile" the comments you make in your code and receive an HTML page

READ MORE about Doxygen here: http://www.doxygen.nl/

- Doxygen template to be posted on Google Drive!

CODE TIP # 2 – THINK BEFORE YOU CODE

Before you dive to coding, think on how you are going to architect your solution:

- What tools does the MCU have to offer to solve our issue?
- Are there ways to use said tools to solve our problem?
- Can we divide the problem into smaller problems?
- What data structures do we need in order to solve our issue/make it easier?

CODE TIP # 2 – THINK BEFORE YOU CODE

Live Example in class – Whiteboard

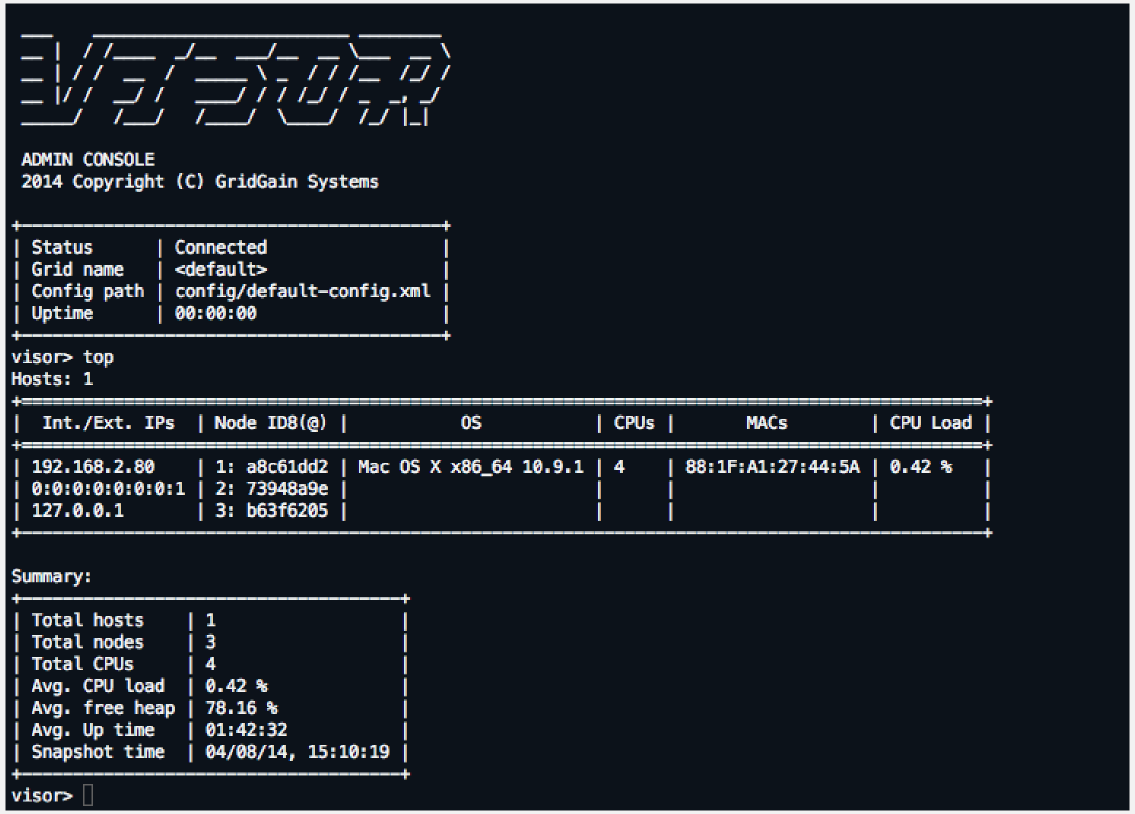
Let's attempt to solve A1- CLI!

A1- CLI

We want the user to be able to type commands to be executed on the MCU

If the user types “help”, they get a list of commands that are possible to be executed on the device.

The user can use a terminal; program, like TERATERM, to write commands and receive the response.



```
 ADMIN CONSOLE
 2014 Copyright (C) GridGain Systems

+------------------------------------------+
| Status      | Connected                  |
| Grid name   | <default>                  |
| Config path | config/default-config.xml  |
| Uptime      | 00:00:00                   |
+------------------------------------------+
visor> top
Hosts: 1
+==============================================================================================+
|  Int./Ext. IPs  | Node ID8(@) |           OS            | CPUs |       MACs        | CPU Load |
+==============================================================================================+
| 192.168.2.80    | 1: a8c61dd2 | Mac OS X x86_64 10.9.1  | 4    | 88:1F:A1:27:44:5A | 0.42 %   |
| 0:0:0:0:0:0:0:1 | 2: 73948a9e |                         |      |                   |          |
| 127.0.0.1       | 3: b63f6205 |                         |      |                   |          |
+----------------------------------------------------------------------------------------------+

Summary:
+--------------------------------------+
| Total hosts    | 1                   |
| Total nodes    | 3                   |
| Total CPUs     | 4                   |
| Avg. CPU load  | 0.42 %              |
| Avg. free heap | 78.16 %             |
| Avg. Up time   | 01:42:32            |
| Snapshot time  | 04/08/14, 15:10:19  |
+--------------------------------------+
visor> 
```